HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

BACHELOR

Herr
Alexander Steinhardt

Untersuchung und Modellierung von Varianten des GLVQ-Algorithmus zur Klassifikation mehrdimensionaler Daten

2010

Fakultät **Mathematik**

BACHELOR

Untersuchung und Modellierung von Varianten des GLVQ-Algorithmus zur Klassifikation mehrdimensionaler Daten

Autor:
Alexander Steinhardt

Studiengang:
Angewandte Mathematik

Seminargruppe:
MA07w1-B

Erstprüfer:
Villmann

Zweitprüfer:
Kästner

Mittweida, 11 2010

I. Vorwort

> *In dieser Welt ist nichts gewiss, außer dem Tod und den Steuern.*
> Benjamin Franklin [11]

Dies schrieb Franklin 1789, einen Tag vor der Französischen Revolution an seinen Freund Leroy[1]. Nun, dieses Zitat werden wohl nur die Wenigsten als motivierend empfinden. Aber gerade zu den Wenigen könnten sich diejenigen zählen, die sich mit dem maschinellen Lernen beschäftigen. Denn in diesem Gebiet der Mathematik und Informatik versucht man, dem Ungewissen Herr zu werden. Dies begegnet einem zum Beispiel bei der Sprach- oder Handschriftenerkennung oder bei Echtzeit- bzw. Simultanübersetzungen in eine Fremdsprache, zum Beispiel denkbar bei einem Telefonat in ein anderes Land. In der Robotik, wo sich Roboter situationsangepasst verhalten sollen, zum Beispiel bei Erkundungen auf anderen Planeten, bei Berechnungen der Kreditwürdigkeit eines Einzelnen und auch bei der medizinischen Diagnostik mithilfe des Computers usw[2]. Warum man sich im Ungewissen bewegt und was man unter dem maschinellen Lernen versteht, wird im ersten Abschnitt geklärt. Dass das Gebiet des maschinellen Lernens recht umfangreich ist, wird schon durch die zahlreichen Anwendungen in den verschiedensten Themengebieten deutlich. Der erste Abschnitt schränkt daher dieses Gebiet auf den Gegenstand unserer Betrachtungen, den GLVQ- Algorithmus, ein. Dieser ist ein spezieller Algorithmus der in einem Teilgebiet des maschinellen Lernens Verwendung findet, wie ebenfalls im ersten Abschnitt erläutert. Ausführlich wird dieser im dritten Abschnitt diskutiert. Der zweite Abschnitt führt die für den GLVQ- Algorithmus zu bewältigende Aufgabe ein und stellt einen Ansatz zur Lösung vor, welcher im dritten Abschnitt umgesetzt wird. Im vierten Abschnitt wird eine Idee zur Verbesserung des GLVQ-Algorithmus vorgestellt, deren Umsetzung beschrieben und die Resultate anhand von Beispielen erklärt werden. Der fünfte und letzte Abschnitt fasst die gesammelten Ergebnisse zusammen.

[1] Vgl. [17]
[2] Vgl. [5], [15]

II. Inhaltsverzeichnis

1 Einführung [5,7]

Klären wir zunächst den Startpunkt unserer Betrachtungen.

Definition 1.1 (Maschinelles Lernen) Sei ein Problem und dessen mathematische Modellierung gegeben. Man versteht unter dem maschinellen Lernen die Ausführung eines, für das Problem konstruierten, Algorithmus auf dem Computer. In diesem Algorithmus wird ein bestimmtes Leistungskriterium anhand von Beispieldaten oder Erfahrungen aus der Vergangenheit in Abhängigkeit der Modellparameter optimiert. Dabei stellt die Anpassung der Modellparameter den Lernprozess dar.

Wir werden im Weiteren das maschinelle Lernen auch einfach nur als Lernen bezeichnen.

Bei mathematischen Aufgabenstellungen werden oft Probleme der Art

$$f : X \to Y \subseteq \mathbb{R}^k \tag{1.1}$$

betrachtet. Dabei ist weder die Vorschrift[3] f noch der Definitionsbereich[4] X von f bekannt. Die Beispieldaten bzw. Erfahrungen liefert der Wertebereich[5] Y, welcher Teilmenge[6] des k- dimensionalen Raumes[7] $\mathbb{R}^k$ ist. Will man zum Beispiel die Gattungen einer Blume anhand von ihren Blattgrößen unterscheiden, so sind lediglich die Messungen der Blattgrößen von Relevanz und zum Beispiel nicht, wie (f) und wo (X) die Blume wächst. Die Unkenntnis von f führt zu der im Vorwort angesprochenen Ungewissheit. Ferner betrachten wir die Beispieldaten als Realisierungen von Zufallsvariablen[8]. In dem soeben konstruierten Beispiel ordnet die Zufallsvariable einer Blume, bestimmter Gattung, einen Vektor[9], der Messungen eines Blattes dieser Blume, zu. Dieser Vektor ist dabei die Realisierung der Zufallsvariablen und zum Beispiel von der Form eines reellwertigen zweidimensionalen Vektors $(\text{Länge des Blattes}, \text{Breite des Blattes})^T \in \mathbb{R}^2$. Einen Vektor der Art kann man auch als Punkt interpretieren[10]. Weiterhin benötigt man für die meisten Probleme oft mehr als eine Zufallsvariable. In diesem Beispiel sind es so viel Zufallsvariablen, wie man Gattungen betrachtet, da man pro Gattung eine Zufallsvariable benötigt. Dieses Beispiel wird im Abschnitt 3.2 konkretisiert. Wir bezeichnen die Beispieldaten bzw. Erfahrungen im Folgenden als die Trainingsdaten bzgl. derer wir lernen.

[3] Vgl. [10], S. 93
[4] Vgl. [10], S. 93
[5] Vgl. [10], S. 93
[6] Vgl. [7], S. 33
[7] Vgl. [10], S. 323f.
[8] Vgl. [15], S. 29f., [18], S. 41f.
[9] Vgl. [9], S. 65
[10] Vgl. [10], S. 323

Die Konstruktion eines für das Lernen benötigten Algorithmus basiert auf gewissen Regeln. Wir wollen diese als Lernregeln bezeichnen. Ferner lasse sich das Modell durch eine Abbildung $\tilde{f}(\Omega|X)$ der zu optimierenden Parametern Ω unter den bekannten Trainingsdaten X beschreiben. Das Ziel ist also die Funktion f durch die erlernte Funktion $\tilde{f}$ zu approximieren.

Soll zum Beispiel ein Computer eine von Hand geschriebene Ziffer erkennen, so benötigt man ein geeignetes Modell mit entsprechenden Parametern, das eine Zifferer-kennung ermöglicht. Dabei ist die Ziffererkennung dass für das Lernen benötigte Leistungskriterium. Man kann dann erst mit dem Computer anhand von gegebenen handgeschriebenen Ziffern lernen. Als nächstes übergibt man dem Modell, mit den optimierten Parametern, eine zu erkennende handgeschriebene Ziffer. Der Computer liefert dann die von ihm erkannte Ziffer. Hierbei sind die Trainingsdaten X die gegebenen handgeschriebenen Ziffern und die Optimierung der Parameter erfolgt über die Ziffererkennung. Ferner bildet die erlernte Funktion mit der Ziffererkennung eine Approximation der handgeschriebenen Ziffern, womit zukünftige handgeschriebene Ziffern durch die erlernte Funktion, unter bestimmten Voraussetzungen, erkannt werden können.

Es gilt zu beachten, dass man eine gemeinsame Verteilung[11] der Zufallsvariablen voraussetzt und somit die Trainingsdaten als auch die zukünftigen Daten von gleicher Form sind. Diese Forderung mache man sich an folgendem Beispiel klar: Eine Zufallsvariable ordne den Bildern handgeschriebener Ziffern eine Matrix[12] zu. Seien die Einträge dieser Matrix in den Positionen Eins, in dem das Bild ein schwarzes Pixel (Bildpunkt) besitzt. Die anderen Positionen der Matrix, also die den weißen Pixeln des Bildes entsprechen, seien Null. Das bietet sich zum Beispiel an, wenn man die Bilder der handgeschriebenen Ziffern im BMP Formt vorliegen hat. Eine Erklärung zu diesem Format findet man in [6]. Man erhält dann nach dem Lernen, bzgl. der handgeschriebenen Ziffern, bestimmte Matrizen. Die Matrizen sind also die Parameter des mathematischen Modells, welches für diese Problemstellung gegeben sei. Das heißt, nach dem Lernen kann man zum Beispiel das Bild einer handgeschriebenen Fünf, durch entsprechenden Vergleich mit den Matrizen, welche man durch das Lernen erhalten hat, mit der Matrix, welche von der Zufallsvariablen für die handgeschriebene Fünf geliefert wird, als Fünf wiedererkennen. Diesen Vergleich wollen wir in diesem Beispiel als Ziffererkennung definieren. Will man nun Gattungen einer Blume anhand ihrer Blattgößen unterscheiden, so wird man die Ziffererkennung nicht nutzen können, da die Verteilung der Zufallsvariablen verschieden sind. Das Urbild der Zufallsvariablen, die einer handgeschriebenen Ziffer eine Matrix zuordnet, sind die Bildpunkte eines Bildes. Das andere Urbild der Zufallsvariablen, welche zu einer gegebenen Blume den Vektor der Messdaten eines Blattes liefert, sind zum Beispiel durch Zahlen repräsentierte Blumen. Demnach sind die Verteilungen verschieden.

[11] Vgl. [18], S. 44, [18], S. 126ff. und S. 143ff.
[12] Vgl. [15], S. 8

Die gemeinsame Verteilung schließt jedoch keine Fehler aus. Hat man zum Beispiel bzgl. handgeschriebener europäischer Ziffern gelernt und will eine handgeschriebene chinesische Ziffer erkennen, wird das in der Regel fehlschlagen. Da die Zufallsvariable eines Bildes einer chinesischen Ziffer eine Matrix liefern wird, welche von anderer Form ist als die erlernten Matrizen der handgeschriebenen europäischen Ziffern. Die Ziffererkennung wird daher zu keiner vernünftigen Zahlenzuordnung führen. Das folgende Bilde verdeutliche die Unterschiede der Ziffern als Gegenüberstellung.

Abbildung 1.1: Europäische Ziffern im Vergleich zu chinesischen Ziffern von 0 bis 9, angepasst aus [1].

Das war wieder ein Beispiel der Klassifikation, welche eine von vielen möglichen Anwendungen des maschinellen Lernens ist. Der GLVQ- Algorithmus selbst wird für Klassifikationsprobleme genutzt.

Allgemein ergeben sich für das maschinelle Lernen zwei Fälle, die man unterscheiden muss:

- Das überwachte Lernen.
- Das unüberwachte Lernen.

Der GLVQ- Algorithmus gehört zum überwachten Lernen.

Definition 1.2 (Überwachtes Lernen) Sei eine Eingabemenge X und eine Ausgabemenge A gegeben. Dann verstehen wir unter dem überwachten Lernen, das Erlernen einer Abbildung $\tilde{f} : \Omega \times X \to A$ von der Eingabe X mit den Parametern Ω auf die Ausgabe A.

Das Charakteristische des überwachten Lernens sind die gegebenen Ausgabewerte. Das heißt für das Beispiel der Ziffererkennung, dass der Algorithmus erkennt, welche Ziffer gerade gelernt wird. Oder im Beispiel der Blumen, zu welcher Gattung die zu einem festen Zeitpunkt betrachteten Messungen gehören. Man kann daher den GLVQ-Algorithmus als „kognitiv“ bezeichnen. Die gegebenen Ausgabewerte ermöglichen eine Behandlung von Problemen der Klassifikation oder Regression. Weiterhin ermöglicht dies während des Lernens eine ständige Kontrolle, wie gut man bisher gelernt hat, in dem man dem Modell die bisher erlernten Parameter und eine Teilmenge der Trainingsdaten, die sogenannten Testdaten, übergibt. Man vergleicht dann die vom Modell gelieferten Werte mit den bekannten Werten aus den Testdaten und weiß, wie gut man bisher gelernt hat. Wenn man das für einen gewissen Zeitraum macht, ergibt sich ei-

ne Genauigkeitsrate (accuracy rate), die angibt, in welchem Zeitschritt wie gut gelernt wurde. Diese wird im Weiteren als Gütekriterium des Lernens verwendet. Ferner kann man auch das Gegenstück betrachten, also den Fehler, dass die Modellwerte nicht mit den Testwerten übereinstimmen. Im Bezug auf die Klassifikation wird dieser Klassifikationsfehler genannt. Diese Überprüfungen erfordern jedoch vor dem Lernen einige Testdaten aus der Menge der Trainingsdaten zu entnehmen. Die Anzahl der zu entnehmenden Daten hängt dabei von der Anzahl der bekannten Trainingsdaten ab und muss also für das jeweilig betrachtete Problem bestimmt werden. Dazu später mehr.

Weiter oben wurde schon angesprochen, dass zur Klassifikation der Algorithmus der allgemeinen lernenden Vektorquantifizierung (GLVQ) gehört. Es gibt noch eine Vielzahl mehr. Wir beschränken uns auf den GLVQ- Algorithmus. Im Folgenden verdeutlichen wir schemenhaft die Themengebiete des maschinellen Lernens, die man passiert, bis man zum GLVQ- Algorithmus gelangt.

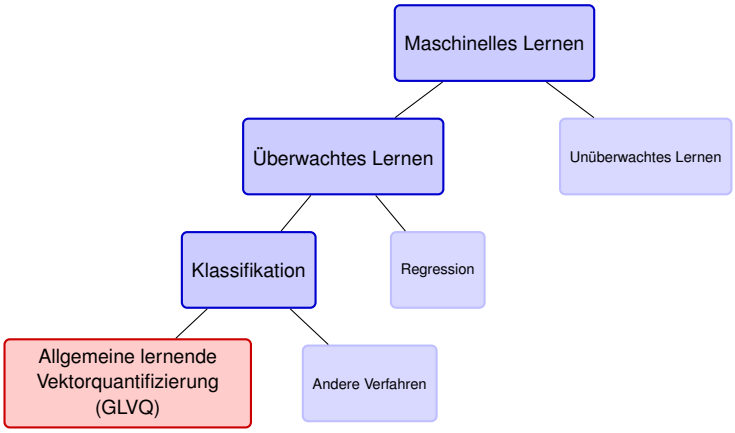

Abbildung 1.2: Einteilung des Themengebietes

1.1 Klassifikation [7]

Die Klassifikation wurde im vorangegangenen Abschnitt für Beispiele herangezogen. Beschäftigen wir uns nun mit der Begriffswelt der Klassifikation.

Definition 1.3 (Klasse) Sei M eine Menge[13] oder Mengensystem[14] und $x \in M$ ein Element der Menge oder des Mengensystems. Dann versteht man unter der Klasse K bzgl. der Klassifikationseigenschaft $\phi(\cdot)$ die Menge aller Elemente $x \in M$, die die Eigenschaft $\phi(x)$ erfüllen:

$$K = \{x \in M | \phi(x)\}$$

Es gibt eine Vielzahl von Möglichkeiten, die Trainingsdaten Y, bzgl. definierter Eigen-

[13] Vgl. [7], S. 15
[14] Vgl. [7], S. 40

schaften, entsprechenden Klassen zuzuordnen. Die Definition der Eigenschaften und somit die Klassenzuordnung der Trainingsdaten muss aus dem zu lösenden Problem hervorgehen oder vorgegeben werden. Betrachte man zum Beispiel die Unterscheidung von Blumengattungen anhand von Blattmessungen und verwende als Klassen die Gattungen einer Blume, so muss für die Klassenzuordnung gegeben sein, welche Messung x zu welcher Gattung mit dem Label $i \in \mathbb{N}$ gehöre.

Definition 1.4 (Label) Sei K eine Menge von Klassen und K_i die i-te Klasse aus K. Ferner besitze K gleich k Elemente, also $|K| = k$. Dann definieren wir die Labelzuordnung der Klassen in K als eindeutige Abbildung in eine Teilmenge der natürlichen Zahlen $\{1,\ldots,k\} \subset \mathbb{N}$:

$$L(K_i) = j \in \{1,\ldots,k\} \subset \mathbb{N}$$

Die Teilmenge $\{1,\ldots,k\}$ der natürlichen Zahlen $\mathbb{N}$ wird Labelmenge genannt und jedes Element $i \in \{1,\ldots,k\}$ der Labelmenge heißt Label.

Eine Menge oder ein Mengensystem liegt dann gelabelt vor, wenn diese(s) in Paaren (x,l) aus Elementen der Menge $x \in M$ und dem Label $l \in \{1,\ldots,n\}$ gegeben ist. Wir wollen eine gelabelte Menge mit dem Index L versehen und schreiben dann $(x,l) \in M_L$ für ein Element aus der gelabelten Menge M_L. Analog gilt dieses für Mengensysteme. Wir verwenden für die folgenden Betrachtungen die Definition der Klasse und Label im Sinne der Mengen. Ferner wollen wir die Ausgabewerte des überwachten Lernens als gelabelt ansehen, um über die Label zuerkennen bzgl. welcher Klasse gerade gelernt wird.

Definition 1.5 (Klassifikation) Seien die Klassen K_L mit $|K_L| = k$ in gelabelter Form gegeben. Dann verstehen wir unter der Klassifikation das überwachte Lernen einer Funktion $\tilde{f} : \Omega \times X \to K_L$ mit den Parametern Ω, den Trainingsdaten X und den Ausgabewerten K_L.

In dem Beispiel der Ziffererkennung des vorangegangenen Abschnittes kann man die Klassen $K_0,\ldots,K_9$ definieren. Zum Beispiel gehöre eine Ziffer x zur Klasse K_i, wenn $x = i$ gilt. Die Klassen sind also gelabelt mit der Labelmenge $\{0,\ldots,9\} \subset \mathbb{N}$. Damit ist eine Klassifikation der Ziffern in sofern möglich, dass man das von der Funktion $\tilde{f}$ gelieferte Label als die erkannte Ziffer interpretiert.

Die zu bewältigende Aufgabe besteht nun darin, in geeigneter Weise eine entsprechende Klassifikation bzgl. vorgegebener Parameter, Trainingsmenge und Klassen zu erlernen.

2 Aufgabe und Lösungsansatz für Klassifikationsalgorithmen [15,16]

Die Aufgabe ist, wie schon festgestellt, das geeignete Lernen einer Funktion $\tilde{f} : \Omega \times X \to K_L$. Dieser Abschnitt konkretisiert, was man unter dem „geeigneten“ Lernen versteht und spezifiziert die Funktion $\tilde{f}(\Omega|X)$ ein wenig mehr. Die Parameter Ω der Funktion $\tilde{f}$ wollen wir als die Menge der Prototypen festlegen.

Definition 2.1 (Prototypen) Man versteht unter einem Prototypen ω einen Punkt $\omega \in \Omega \subseteq \mathbb{R}^k$. Die Menge aller Prototypen wollen wir mit Ω bezeichnen.

Der Prototyp befindet sich nach (1.1) im selben Raum wie die Punkte der Trainingsmenge. Da die Prototypen die zu optimierenden Parameter unserer Klassifikation sind, kann die Aufgabe so umformuliert werden, dass eine Optimierung der Prototypen gesucht wird, um eine bestmöglichste Genauigkeitsrate bzgl. der Trainingsdaten X zu erzielen. Es können dabei auch mehrere Prototypen zu derselben Klasse gehören. Zum Beispiel können die n Prototypen $\omega_1, \ldots, \omega_n$ zur Klasse 1 gehören, womit $(\omega_i, 1) \in \Omega_L$ für $i = 1, 2, \ldots, n$ gilt.

Konkret stellt sich also die Frage, wie die Prototypen zu lernen sind? Eine Antwort liefert Kushner und Clark in [16] unter Verwendung des Robbins-Monro-Prozesses

$$X_{n+1} = X_n - a_n[Y_n - \alpha] = X_n - a_n[f(X_n) - \alpha + \varepsilon_n] \tag{2.1}$$

mit X_n als die n-te Approximation der Zufallsvariablen X, $a_n \in \{a_k\}_{k=1}^{\infty}$ als das n-te Glied der Folge $\{a_k\}_{k=1}^{\infty}$, $f(X_n)$ als die den n-ten Beispielwert Y_n liefernde Funktion, α als gegebener Durchschnittswert von $f(X)$ der durch entsprechende Justierung von X erreicht wird und ε_n als der Fehlerterm von $f(X_n)$. Man nennt ε_n auch Störung der Funktion $f(X_n)$. Etwas vereinfacht lässt sich die Gleichung (2.1) darstellen als

$$X_{n+1} = X_n + a_n h(X_n) + a_n \beta_n \tag{2.2}$$

mit $h(X_n) = \alpha - f(X_n)$ und $\beta_n = (-\varepsilon_n)$. In ähnlicher Form verwendet Kohonen diese Gleichung in [15] auf Seite 43. Die Konvergenz für ein Problem im $\mathbb{R}^k$ sichern Kushner und Clark unter den Bedingungen[15]:

B1 $h : \mathbb{R}^k \to \mathbb{R}^k$ sei stetig

B2 $\{\beta_n\}$ sei eine beschränkte Folge von Zufallsvariablen im $\mathbb{R}^k$ für die $\beta_n \to 0$ mit Wahrscheinlichkeit 1, für $n \to \infty$, gilt.

B3 $\{a_n\}_{n=1}^{\infty}$ sei eine Folge positiver reellwertiger Zahlen für die $a_n \to 0$ für $n \to \infty$,

[15] Vgl. [16]

$\sum_{n=1}^{\infty} a_n = \infty$ und $\sum_{n=1}^{\infty} a_n^2 < \infty$ gilt.

Definition 2.2 (Lernrate) Wir nennen eine positiv reellwertige Folge $\{a_n\}_{n=1}^{\infty}$ Lernrate, wenn sie die Bedingung B3 erfüllt.

Wir verwenden eine etwas einfachere Form der Gleichung (2.2) und bezeichnen diesen Vorgang als Adaption:

$$\omega_{n+1} = \omega_n + \eta_n \Delta\omega \tag{2.3}$$

Hierbei wurde sich unter der obig definierten Aufgabenstellung auf Prototypen $\omega \in \Omega$ bezogen und η sei eine Lernrate. Wir sprechen also im Folgenden von der Adaption der Prototypen $\omega \in \Omega$. Der Index n bezeichnet den aktuellen Lernschritt, da die Adaption „schrittweise“ erfolgt. Diese Schritte bauen aufeinander auf und deshalb nennt man den GLVQ- Algorithmus Markovsch in Anlehnung an den Markovprozess[16] wobei die Zustände die aktuellen Prototypen und den zufällig gezogenen Trainingspunkt umfassen. Genaueres folgt am Ende dieses Abschnittes.

Es gilt nun die zu optimierende Funktion $\tilde{f}(\Omega|X)$ etwas mehr zu spezifizieren. Nach Aufgabenstellung ist diese Funktion so zu wählen, dass man für optimierte Prototypen eine bestmögliche Genauigkeitsrate bzgl. einer Eingabe X erreicht. Es stellt sich nun die Frage, wann wird die Genauigkeitsrate bestmöglich?

Lemma 2.3 *Die Genauigkeitsrate ist genau dann maximal, wenn der Klassifizierungsfehler minimal ist.*

Beweis: Dieser Sachverhalt ist per Definition des Klassifizierungsfehlers klar. □

Man könnte also $\tilde{f}(\Omega|X)$ als Klassifizierungsfehler wählen und minimieren. Man spricht oft, zum Beispiel im Bereich der „Self- Organizing Maps“ des maschinellen Lernens, von der zu minimierenden Energiefunktion[17]. Für den GLVQ- Algorithmus wird im nächsten Abschnitt eine Energiefunktion E definiert und deren Minimierung erfolgt ebenfalls über den stochastischen Gradientenabstieg. Dies ist ein wesentlicher Vorteil gegenüber dem Klassifizierungsfehler, da dieser nicht abgeleitet werden kann.

Definition 2.4 (Gradient) Unter dem Gradienten ∇_ξ einer Funktion E versteht man den Vektor der partiellen Ableitungen[18] dieser Funktion nach $\xi_1, \ldots, \xi_n$:

$$\nabla_\xi(E) = \left(\frac{\partial}{\partial \xi_1}, \ldots, \frac{\partial}{\partial \xi_n}\right)^T \cdot E = \left(\frac{\partial E}{\partial \xi_1}, \ldots, \frac{\partial E}{\partial \xi_n}\right)^T$$

Nach [10] zeigt der Gradient einer Funktion $E(x)$ im Punkt x immer in Richtung des

[16] Vgl. [14], S. 348
[17] Vgl. [15]
[18] Vgl. [10], S. 349

stärksten Anstiegs dieser Funktion $E(x)$. Man erhält in der entgegengesetzten Richtung, die Richtung des steilsten Abstiegs $-\nabla_\xi E(x)$ der Funktion $E(x)$ im Punkt x. Diese Richtung wird für Minimierungsprobleme ohne Nebenbedingung verwendet und man spricht in diesem Zusammenhang vom Gradientenabstieg bzw. vom Gradientenverfahren. Es folgt ein abgewandeltes Beispiel des Gradientenabstiegs nach [8] anhand der Funktion $g(x,y) = -(0.5+\sin(y))(1+\cos(2x))$, wobei sich auf das Intervall $-1.5 \leq x \leq 1.5$ und $-1.0 \leq y \leq 3.5$ beschränkt wurde.

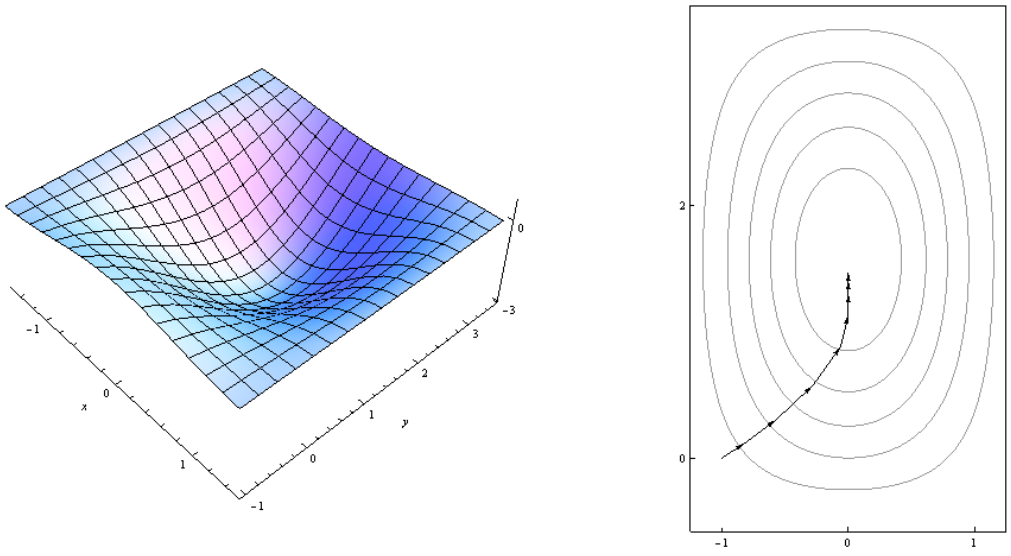

Abbildung 2.1: Graph und Gradientenabstieg der Funktion $g(x,y) = -(0.5+\sin(y))(1+\cos(2x))$ im Bereich von $-1.5 \leq x \leq 3.5$ und $-2 \leq y \leq 4$

Der Gradientenabstieg ist im rechten Bild dargestellt. Die Ellipsen dieser Abbildung stellen die Höhenlinien bzw. Niveaulinien[19] der Funktion $g(x,y)$ dar. Der Abstieg ist als Sequenz von Pfeilen zu erkennen. Die Sequenz entsteht aus der mit dem Gradientenabstieg einhergehenden Iteration[20]. Dabei ist nicht garantiert, dass dieses Verfahren gegen das globale Optimum konvergiert. Ergänzen wir nun die stochastische Herangehensweise zu dem Gradienten und gelangen zum stochastischen Gradientenabstieg[21].

Definition 2.5 (Stochastischer Gradientenabstieg) Sei $\nabla_\xi E = \left(\frac{\partial E}{\partial \xi_1}, \ldots, \frac{\partial E}{\partial \xi_n}\right)^T$ der Gradient der Funktion E. Dann nennen wir den Gradientenabstieg stochastisch, wenn wir diesen N-mal zufällig von verschiedenen Startpunkten aus starteten. Dabei sein $N \in \mathbb{N}$ beliebig aber fest.

Durch den stochastischen Gradientenabstieg und den damit verbundenen zufälligen un-

[19] Vgl. [20], S. 270
[20] Vgl. [12], S. 67ff.
[21] Vgl. [19]

terschiedlichen Startwerten des Gradientenverfahrens erreicht man eine höhere Wahrscheinlichkeit, das globale Minimum einer Funktion zu erreichen. Betrachtet man zum Beispiel wieder die Funktion $g(x,y) = -(0.5+\sin(y))(1+\cos(2x))$, aber dieses Mal im Intervall von $-1.5 \leq x \leq 5.0$ und $-3.0 \leq y \leq 4.5$ unter Verwendung von sechs zufälligen Startpunkten. Dann gelangt man zu folgender Grafik.

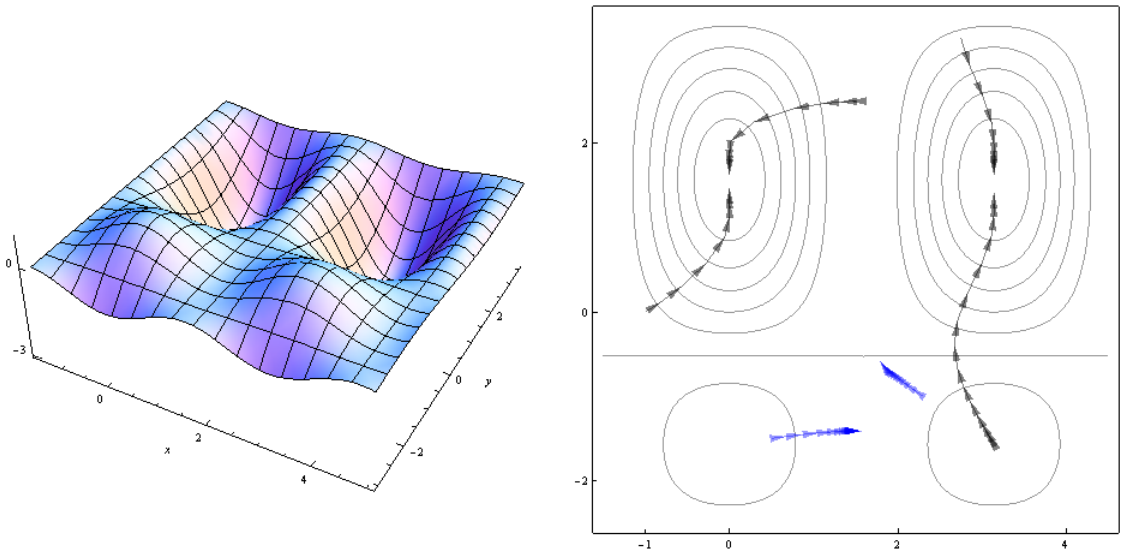

Abbildung 2.2: Graph und stochastischer Gradientenabstieg der Funktion $g(x,y) = -(0.5+\sin(y))(1+\cos(2x))$ im Bereich von $-1.5 \leq x \leq 5.0$ und $-3.0 \leq y \leq 4.5$

Man erkennt die grauen Gradientenabstiege, die in einem der zwei globalen Minima enden und man erkennt die blauen Gradientenabstiege, die in einem lokalem Minima enden.

Ferner wird in dem GLVQ- Algorithmus das stochastische Verfahren des „online“ Lernens verwendet. Das wird im nächsten Abschnitt noch hervorgehoben.

Definition 2.6 (Online Lernen) Ein Lernverfahren heißt online, wenn man für die zu erlernende Funktion $\tilde{f}$ nicht die kompletten Trainingsdaten verwendet, sondern bzgl. eines für jeden Lernschritt zufällig entnommenen Elementes $x \in X$ der Trainingsdaten gelernt wird.

Klären wir nun noch, warum der GLVQ- Algorithmus Markovsch ist. Durch das online Lernen bezieht man sich auf einen einzelnen zufällig gezogenen Trainingspunkt pro Lernschritt. Nach Bedingung B2 und B3 ist die Konvergenz erst nach theoretisch unendlich vielen Lernschritten gesichert. Ferner bauen die Lernschritte aufeinander auf, da jeder Lernschritt den Prototypen adaptiert und somit einen neuen Startpunkt bzgl. der Prototypen für den nächsten Lernschritt festlegt. Dieses Verhalten nennt man Markovsch im Sinne des Markovprozesses.

Die durch das Lernen konstruierte Folge $\{\omega_n\}_{n=1}^{\infty}$ von Prototypen entspricht der in dem Robbins-Monro-Prozess (2.1) konstruierten Folge von Approximationen der Zufallsvariablen X, es gilt also $\{X_n\} = \{\omega_n\}$. Im Allgemeinen hat man für den Lernprozess jedoch nicht unendlich viel Zeit zur Verfügung, so dass man nach N Schritten das Lernverfahren abbricht und lediglich eine Approximation der Klassifikation erhält. Ferner wirkt sich das online Lernen auch auf den stochastischen Gradientenabstieg aus. Da sich nur noch auf ein Element bezogen wird, besitzt der stochastische Gradientenabstieg nur noch einen Approximationsschritt und bekommt im nächsten wieder zufällig einen neuen Startpunkt, von dem aus wieder ein Approximationsschritt ausgeführt wird. Bezieht man sich auf das Beispiel der Funktion $g(x,y) = -(0.5+\sin(y))(1+\cos(2x))$ im Bereich von $-1.5 \leq x \leq 5.0$ und $-3.0 \leq y \leq 4.5$, lässt sich dies dadurch verdeutlichen, dass die Sequenz der stochastischen Gradientenabstiege jeweils nur noch einen Pfeil besitzt. Betrachte man dazu folgende Grafik mit 15 stochastischen „online“ Gradienten.

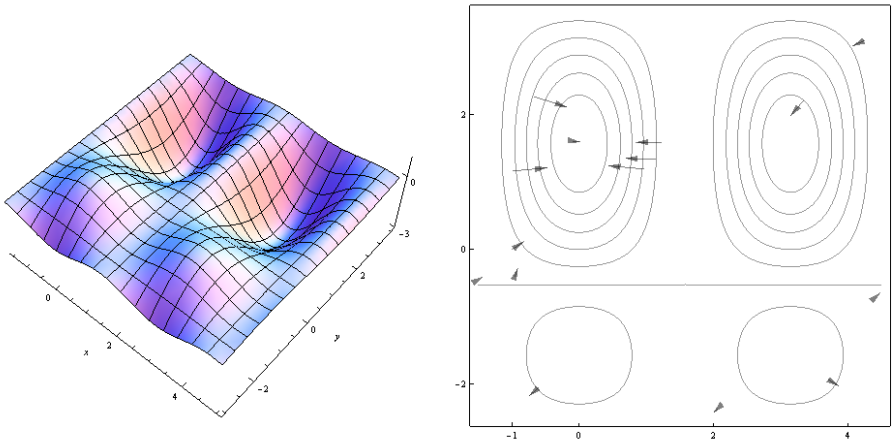

Abbildung 2.3: Graph und stochastischer online Gradientenabstieg der Funktion $g(x,y) = -(0.5+\sin(y))(1+\cos(2x))$ im Bereich von $-1.5 \leq x \leq 5.0$ und $-3.0 \leq y \leq 4.5$ für 15 stochastische „online“ Gradienten.

Wie das im Einzelnen in den GLVQ- Algorithmus eingeht wird im nächsten Abschnitt erläutert.

3 Herleitung des GLVQ- Algorithmus [13]

Dieser Abschnitt wird nun aufzeigen, wie man die Energiefunktion E minimiert

$$E \to \min$$

und wie dies zu entsprechenden Lernregln für den GLVQ- Algorithmus führt.

Im ersten Abschnitt wurde in (1.1) bereits festgestellt, dass wir uns in einem k-dimensionalen Raum bewegen. Ferner wurde im zweiten Abschnitt definiert, dass die Prototypen ω ebenfalls in diesem Raum liegen. Wir wollen als nächstes einen Abstand zwischen den Punkten dieses Raumes festlegen.

Definition 3.1 (Quadratischer euklidischer Abstand) Seien $x, y \in \mathbb{R}^k$ zwei Punkte aus dem k-dimensionalen Raum $\mathbb{R}^k$. Dann verstehen wir unter dem quadratischen euklidischen Abstand $d(x,y)$ folgenden Ausdruck:

$$d(x,y) = \sum_{i=1}^{k} (x_i - y_i)^2 \tag{3.1}$$

Im Weiteren interessieren uns spezielle Abstände zwischen den gelabelten Trainingsdaten X_L und den gelabelten Prototypen Ω_L. Zum Einen der quadratische euklidische Abstand eines Punktes $(x,i) \in X_L$ zum nächsten Prototypen $(\omega^+, i) \in \Omega_L$ der zur selben Klasse i wie der Punkt x gehört. Wir bezeichnen diesen Abstand als Gewinnerabstand $d^+(x, \omega^+)$ eines Punktes x zum Prototypen ω^+. Zum Anderen der quadratische euklidische Abstand eines Punktes $(x,j) \in X_L$ zum nächsten Prototypen $(\omega^-, k) \in \Omega_L$, der nicht zur selben Klasse wie der Punkt x gehört, also $k \neq j$. Wir bezeichnen diesen Abstand als Verliererabstand $d^-(x, \omega^-)$ eines Punktes x zum Prototypen ω^-. Wir schreiben kurz $d^+(x)$ für den Gewinnerabstand und $d^-(x)$ für den Verliererabstand.

Unter Zuhilfenahme dieser Abstände lässt sich folgende Funktion definieren:

$$1^i_{WTA}(x) = \begin{cases} 1 & \text{falls } d^+(x, \omega_i) \text{ ist minimal} \\ 0 & \text{sonst} \end{cases} \tag{3.2}$$

Das ist die sogenannte „Winner-Takes-All“ Funktion des i-ten Prototypen. Diese ermöglicht eine Klassifikation der Art:

$$\tilde{f} : \Omega_L \times X \to K_L : ((\omega_j, i), x) \mapsto i \cdot 1^j_{WTA}(x) \tag{3.3}$$

Es werden also all diejenigen Datenpunkte $x \in X$ einer entsprechenden Klasse i zugeordnet, die bzgl. des Abstandes $d^+(x)$ den geringsten Abstand zu dem gelabelten

Prototypen $(\omega_j, i) \in \Omega_L$ besitzen. Aus dieser Definition wird klar, dass man die Prototypen als Repräsentanten der zu ihnen bzgl. eines Abstandes und bzgl. aller anderen Prototypen nahest gelegen Datenpunkte $x \in X$ bezeichnen kann. Somit erreicht man eine Trennung der Klassen durch Prototypen und letztendlich durch den GLVQ- Algorithmus. Für diese Klassifikation sind die Label der Trainingsmenge nicht erforderlich und können einfach ignoriert werden. Für den Test gilt, dass diejenigen Datenpunkte aus der Testmenge, die einer anderen Klasse zugeordnet werden, als die ihnen Gegeben ist, wie in Abschnitt 1 schon beschrieben, fehlklassifiziert sind.

Damit lässt sich die in Abschnitt 2 definierte Aufgabe umformulieren. Man sucht also eine bestmögliche Repräsentation der Daten durch so wenig wie möglich Prototypen. Wir hatten schließlich im vorangegangen Abschnitt festgestellt, dass auch mehrere Prototypen pro Klasse möglich sind. Somit ist es auch möglich, dass man jeden Punkt, der zur entsprechenden Klasse gehört, durch einen Prototypen repräsentiert. Das ist aber in sofern ungünstig, dass man zum Einen keine Prototypen hätte erst einführen müssen, da ja jeder Punkt äquivalent zu einem Prototypen ist. Zum Anderen die Aussagekraft wesentlich geringer ist, da die Datenpunkte selbst nicht mehr gelernt werden brauchen. Es ist ja schon jeder Prototyp eindeutig bestimmt, und somit wird das Verhalten der unbekannten Funktion f nicht approximiert, was aber gerade das Ziel des Lernens ist, siehe Abschnitt 1. Weiterhin lässt diese Variante keine Interpretation für zukünftige ungelabelte Daten zu, da ja nicht gelernt wurde. Man will also mit so wenig wie möglich Prototypen auskommen um die Funktion f so gut wie möglich zu approximieren. Wie viel konkret genommen werden sollten ist vom betrachteten Problem abhängig.

Ferner haben Sato und Yamada die folgende Energiefunktion für den GLVQ- Algorithmus konstruiert[22]:

$$\hat{E} = \frac{1}{2} \sum_j \frac{d^+(x_j) - d^-(x_j)}{d^+(x_j) + d^-(x_j)} \tag{3.4}$$

Dabei ist x_j der j-te Trainingspunkt aus der Trainingsmenge X. Der Nenner der einzelnen Terme der Summe sorgt dafür, dass die Terme im Intervall $[-1, 1]$ liegen. Der Zähler ist also wesentlich für die Lösung unserer Aufgabe der bestmöglichen Repräsentation, da dieser für $\omega^+ \to x_j$ negativ wird. Denn für den Abstand gilt, $d^+(x) \to 0$ für $\omega^+ \to x$ und in diesem Fall reduziert sich der Summand auf

$$\frac{-d^-(x_j)}{d^-(x_j)} = -1.$$

Die Summanden werden also kleiner um so besser der jeweilige Prototyp ω^+ die Trainingsdaten repräsentiert. Dies führt zu der entsprechenden Abnahme der Energiefunktion und somit zum Minimum. Das ist es also, was es mit der Adaption zu erreichen gilt. Weiterhin gilt, dass wenn die Energiefunktion klein wird auch der Klassifikationsfehler klein wird, da der Abstand $d^+(x)$ nach (3.3) und somit nach (3.2) als Minimum in die

[22] Vgl. [21]

Klassifikation eingeht. Das war schließlich auch der Gedanke von Sato und Yamada in [21], wo die Kosten- bzw. Energiefunktion (3.4) im Zusammenhang mit dem stochastischen Gradientenabstieg eingeführt wurde. Und somit löst die Minimierung der Energiefunktion (3.4) die in Abschnitt 2 und die in diesem Abschnitt äquivalente formulierte Aufgabenstellung.

Wie im Abschnitt 2 definiert wurde, gehört der GLVQ- Algorithmus zu den online Lernverfahren. Das führt dazu, dass wir die Energiefunktion E nicht für alle Trainingsdaten $x \in X$ betrachten, sondern lediglich für einen Einzelnen zufällig gezogenen Datenpunkt $x \in X$. Das heißt, die Summe in der Energiefunktion E reduziert sich auf einen einzelnen Summanden bzgl. des zufällig gezogenen Datenpunktes $x \in X$:

$$E(x) = \frac{1}{2} \cdot \frac{d^+(x) - d^-(x)}{d^+(x) + d^-(x)}$$

Da ferner die Minimierung von $\hat{E}$ und somit auch von E über den stochastischen Gradientenabstieg erfolgen soll, betrachten wir zu nächst den Gradienten von E bzgl. der Prototypen ω^+ und ω^-.

$$\frac{\partial E}{\partial \omega^+} = \triangle \omega^+ = \frac{1}{2} \cdot \frac{d^-(x_j)}{\left(d^+(x_j) + d^-(x_j)\right)^2} \cdot \frac{\partial d^+(x_j)}{\partial \omega^+} \tag{3.5}$$

$$\frac{\partial E}{\partial \omega^-} = \triangle \omega^- = \frac{1}{2} \cdot \frac{d^+(x_j)}{\left(d^+(x_j) + d^-(x_j)\right)^2} \cdot \frac{\partial d^-(x_j)}{\partial \omega^-} \tag{3.6}$$

$$\nabla_{(\omega^+, \omega^-)^T} E = \begin{pmatrix} \frac{\partial E}{\partial \omega^+} \\ \frac{\partial E}{\partial \omega^-} \end{pmatrix} = \begin{pmatrix} \triangle \omega^+ \\ \triangle \omega^- \end{pmatrix} \tag{3.7}$$

Dabei ist nach Gleichung (3.1) $\frac{\partial d^+(x_j)}{\partial \omega^+} = 2(x_j - \omega^+)$ und $\frac{\partial d^-(x_j)}{\partial \omega^-} = 2(x_j - \omega^-)$. Es ergibt sich für die Adaption nach Gleichung (2.3) folgende Form:

$$\omega^{\pm}_{n+1} = \omega^{\pm}_n \pm \eta_n \triangle \omega^{\pm}$$

Dies lässt sich einzeln schreiben als:

$$\omega^+_{n+1} = \omega^+_n + \eta_n \triangle \omega^+ \tag{3.8}$$

$$\omega^-_{n+1} = \omega^-_n - \eta_n \triangle \omega^- \tag{3.9}$$

Man erhält also eine Adaption der Prototypen durch zwei Gleichungen. Wir wollen dies im Weiteren als Gradientenschritt- Adaption bezeichnen. Wie im vorangegangenen Abschnitt erläutert, wird für den Gradientenschritt der stochastische Gradientenabstieg genutzt, wobei dieser noch durch die Lernrate η_n gedämpft wird. Ferner beschreiben die zwei Gleichungen die wesentliche Idee der lernenden Vektorquantifizierung (LVQ)[23]. Man adaptiert in der ersten Gleichung den Prototypen, der zur selben Klasse gehört wie

[23] Vgl. [13]

der zufällig gezogenen Datenpunkt selbst. Dieser Prototyp wird von dem zufällig gezogenen Datenpunkt durch $\eta_n \triangle \omega^+ = \eta_n \frac{d^-(x_j)}{\left(d^+(x_j)+d^-(x_j)\right)^2} \cdot (x_j - \omega^+)$ angezogen. Das ist schon Bestandteil der online Vektorquantifizierung. Das Charakteristische für den LVQ-Algorithmus ist also, dass man mit der zweiten Gleichung (3.9) den Prototypen, der ungünstig liegt, ebenfalls adaptiert. Dies geschieht aber in entgegengesetzter Richtung, also der Prototyp wird mit $-\eta_n \triangle \omega^- = -\eta_n \frac{d^+(x_j)}{\left(d^+(x_j)+d^-(x_j)\right)^2} \cdot (x_j - \omega^-)$ abgestoßen. Die Adaption bzgl. eines zufälligen Datenpunktes ist theoretisch unendlich oft zu wiederholen, siehe Abschnitt 2. Wir beschränken uns jedoch auf eine Approximation durch N Lernschritte.

Betrachten wir abschließend den Algorithmus, der sich aus dem Vorangegangenem ergibt.

GLVQ:
wähle die Anzahl an Iterationen $k_{max} = N$ und setze $k = 0$
initialisiere alle Prototypen $\omega_0 \in \Omega$ zufällig
while $k < k_{max}$ **do**
(S1) wähle ein $x \in X$ zufällig.
(S2) ermittle den zu x gehörigen Gewinnerprototypen ω^+ und Verliererprototypen ω^- mit $\omega^{\pm} \in \Omega$
(S3) berechne den Gewinnerabstand $d^+(x)$ und Verliererabstand $d^-(x)$
(S4) adaptiere ω^+_{k+1} nach Gleichung (3.8) und adaptiere ω^-_{k+1} nach Gleichung (3.9)
(S5) erhöhe k mit $k = k+1$
end while

3.1 Spezielle Lernraten [20]

Dieser kurze Abschnitt erläutert eine geeignete Wahl der Lernrate. Dazu bedienen wir uns einiger Sätze aus der Analysis, welche in [20] beschrieben sind.

Satz 3.2 *Sei $\{a_n\}$ eine Folge[24]. Konvergiert die Reihe[25] $\sum_{n=1}^{\infty} a_n$, dann ist* $\lim_{n\to\infty} a_n = 0$.

Beweis: Siehe [20] Seite 68. □

Satz 3.3 *Sei $\{a_n\}$ eine Folge für die $a_1 \geq a_2 \geq \cdots \geq 0$ gilt. Dann konvergiert die Reihe $\sum_{n=1}^{\infty} a_n$ genau dann, wenn die Reihe*

$$\sum_{k=0}^{\infty} 2^k a_{2^k} = a_1 + 2a_2 + 4a_4 + 8a_8 + \ldots$$

[24] Vgl. [20], S. 29
[25] Vgl. [20], S. 67

konvergiert.

Beweis: Siehe [20] Seite 70. □

Es folgt der für die Lernrate wichtige Satz.

Satz 3.4 *Sei $n \in \mathbb{N}$ dann gilt, dass $\sum \frac{1}{n^p}$ für $p > 1$ konvergiert und für $p \leq 1$ divergiert.*

Beweis: Die Divergenz aus $p \leq 0$ folgt aus Satz 3.2 durch Negation der Aussage. Sei $p > 0$ und unter Anwendung des Satzes 3.3 folgt, dass wenn

$$\sum_{k=0}^{\infty} 2^k \cdot \frac{1}{2^{kp}} = \sum_{k=0}^{\infty} 2^{(1-p)k}.$$

konvergiert auch $\sum \frac{1}{n^p}$ konvergiert. Es ist $2^{1-p} < 1$ genau dann, wenn $1 - p < 0$ gilt. Unter Anwendung der geometrischen Reihe[26] erhält man

$$\sum_{k=0}^{\infty} 2^{(1-p)k} = \frac{1}{1-2^{1-p}}.$$

Es folgt also die Aussage des Satzes. □

Aus dem Beweis wird klar, dass die Aussage des Satzes 3.4 noch gilt, wenn die Reihe mit einer positiven reellen Zahl $c \in \mathbb{R}^+$ multipliziert wird.

Korollar 3.5 *Sei $n \in \mathbb{N}$ und $c \in \mathbb{R}^+$ dann gilt, dass $\sum \frac{c}{n^p}$ für $p > 1$ konvergiert und für $p \leq 1$ divergiert.*

Damit erhält man eine geeignete Lernrate durch Wahl der Folge $\{\frac{c}{n}\}$ von positiven rationalen Zahlen mit $n \in \mathbb{N}$ und $c \in \mathbb{R}$ unter geeigneter Wahl von p, so dass die Bedingung B3 aus Abschnitt 2 erfüllt ist. Das heißt es muss $\sum_{n=1}^{\infty} \frac{c}{n^p} = \infty$ und $\sum_{n=1}^{\infty} \frac{c^2}{n^{p^2}} = \sum_{n=1}^{\infty} \frac{\tilde{c}}{n^{2p}} < \infty$ mit $\tilde{c} \in \mathbb{R}$ gelten. Dies gilt nach Korollar 3.5 und Satz 3.4 für $\frac{1}{2} + \varepsilon < p < 1$ mit $\varepsilon > 0$.

3.2 Beispiel - Iris Blumen

Der Algorithmus **GLVQ** wurde in MATLAB anhand des Iris- Blumendatensatzes umgesetzt. Klären wir zunächst, was es mit dem Iris- Datensatz auf sich hat. Den Irisdatensatz an sich erhält man unter [3]. Dieser enthält Messungen zu drei Gattungen der Irisblume, auch bekannt als „Schwertlilie“. Die drei Gattungen sind:

- Setosa
- Versicolor

[26] Vgl. [20], S. 69

- Virginica

Dabei wurden vier Messungen an jeder Blume vorgenommen. Es wurden Länge und Breite des Blütenblattes (petal) und Länge und Breite des Kelchblattes (sepal) gemessen. Das Bild verdeutlicht, welche Blätter der Irisblume gemeint sind.

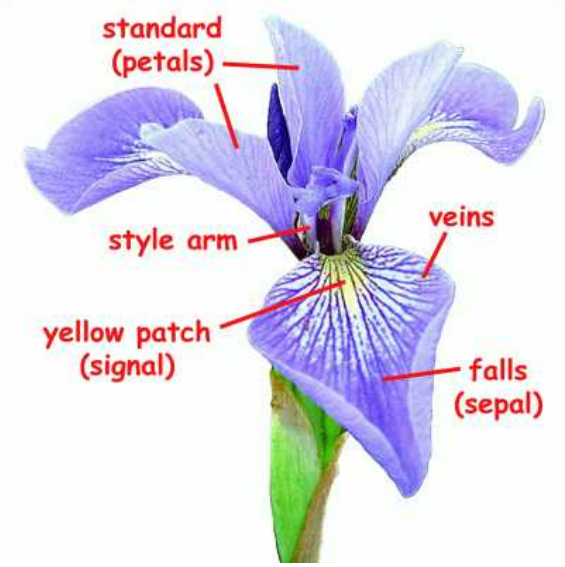

Abbildung 3.1: Irisblume mit Blatterklärung aus [2]

Der Datensatz besteht aus 50 Messungen pro Gattung. Es liegen also 150 Messungen vor. Sowie drei Klassen, die wir wie folgt festlegen:

- Setosa sei Klasse 1
- Versicolor sei Klasse 2
- Virginica sei Klasse 3

Um den Lernerfolg zu messen, wurden pro Gattung zufällig (d.h. in MATLAB über die rand[27] Funktion) 15 Datensätze für Testzwecke entnommen, die sogenannten Testdaten. Es verbleiben somit 35 Messungen pro Klasse. Da das nicht genug Daten sind um eine Funktion vernünftig zu erlernen, wird eine sogenannte Kreuzvalidierung vorgenommen. Das heißt, wir wiederholen den Vorgang einige Male und mitteln das Ergebnis für eine bessere Aussagekraft. Das Gütekriterium ist, wie schon erwähnt, die Genauigkeitsrate. Wir haben uns aufgrund der Kreuzvalidierung entschieden, den Vorgang vier mal zu wiederholen. Ferner verwendeten wir einen Prototypen pro Klasse und folgende Lernrate:

$$\eta_n = \frac{0.2 - 0.0001}{n^{\frac{1}{2}+0.005}}$$

Die Anzahl der Lernschritte wurde auf $N = 5000$ begrenzt. Man gelangt zu folgender Grafik als Ergebnis.

[27] Vgl. [4]

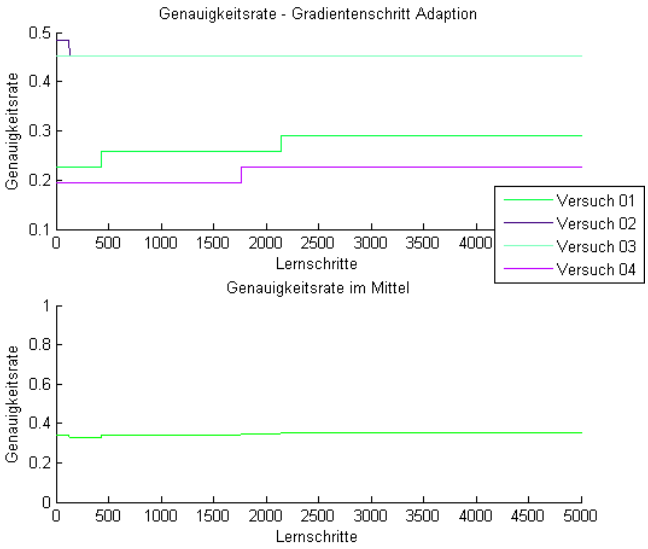

Abbildung 3.2: Genauigkeitsrate für den GLVQ- Algorithmus mit 5000 Lernschritten und Lernrate η_n bzgl. Gradientenschritt- Adaption.

Das Ergebnis ist natürlich von der Initialisierung abhängig, das heißt, der GLVQ- Algorithmus ist sensitiv bzgl. der Initialisierung. In diesem Bild sind aber lediglich die Anfänge des Lernens in den Versuchen 01 und 04 zu erkennen. Um das Lernverhalten besser zu sehen, müsste N wesentlich größer gewählt werden. Man kann nun aber auch versuchen die Lernrate zu verändern, da diese das Lernverhalten deutlich beeinflusst, also ist der GLVQ- Algorithmus, wie jeder Lernalgorithmus, auch sensitiv bzgl. der Lernrate. Wählt man zum Beispiel

$$\alpha_n = \frac{0.2 - 0.0001}{n^{\frac{1}{8}}}$$

für die Lernrate so gelangt man zu folgender Grafik.

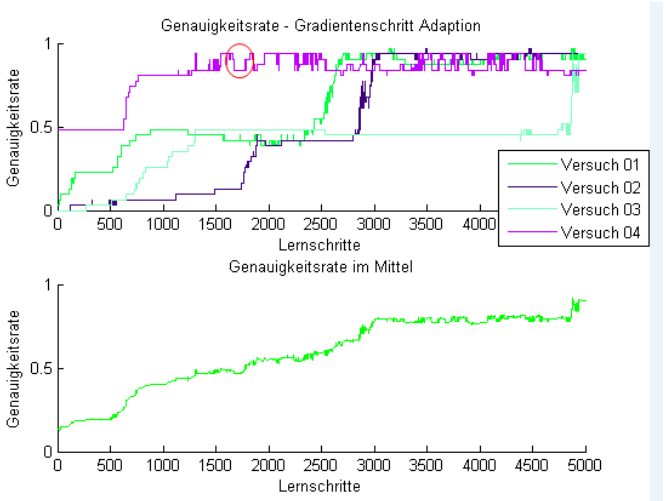

Abbildung 3.3: Genauigkeitsrate für den GLVQ- Algorithmus mit 5000 Lernschritten und Lernrate α_n bzgl. Gradientenschritt- Adaption.

Man sieht also eine deutliche Verbesserung des Lernverhaltens, was den Einfluss der Lernrate unterstreicht. Hierbei erfüllt nun jedoch die verwendete Lernrate α_n die Bedingung $\sum_{n=1}^{\infty} \alpha_n^2 < \infty$ nicht. In der Praxis ist es durchaus üblich sich nicht immer an die Bedingungen $\sum_{n=1}^{\infty} \alpha_n^2 < \infty$ zu halten. Dennoch kann dies, wie gesehen, zu einem erfolgreichen Lernverhalten führen. Das wiederum ist dadurch begründet, dass diese Bedingungen dafür sorgen, dass die im Robbins-Monro-Prozesses (2.1) auftretenden Störungen ε_n im Unendlichen keinen Einfluss mehr auf die betrachtete Funktion ausübt[28]. Wenn nun von vornherein keine großen Störungen ε_n vorliegen, kann diese Bedingung auch weggelassen werden. Dabei ist zu beachten, dass man in der Regel nicht weiß, wie groß die Störungen sind und es in der Praxis nur ausprobieren kann, ob eine andere Lernrate zum Erfolg führt.

In der obigen Grafik lässt sich im unteren Bild gut erkennen, dass die Genauigkeitsrate im Großen und Ganzen ein monoton steigendes Verhalten liefert. Bei dieser können auch Schwankungen auftreten. Eine ist im Versuch 04 durch einen roten Kreis im oberen Bild der Grafik verdeutlicht. Diese Schwankungen treten aufgrund des stochastischen Gradientenabstieges im Zusammenhang mit dem online Lernen auf, in dem der zufällig gezogene Trainingspunkt sich durch die Adaption ungünstig auf die Klassifizierung auswirkt. Ferner kann es auch passieren, dass die Genauigkeitsrate, nach erreichen eines Höchstwerts, etwas absinkt und sich auf ein etwas niedrigeres Niveau einpendelt. Das folgt daher, dass man schließlich nicht abbricht, sobald man eine gute Genauigkeitsrate erreicht hat. Durch das weiter Lernen gelangt man zu schlechteren Genauigkeitswerten, da die Prototypen weiter durch andere zufällig gewählte Datenpunkte adaptiert werden, was sich wiederum ungünstig auf die aktuelle Genauigkeitsrate auswirken kann.

Noch schnellere und bessere Ergebnisse liefert der GLVQ- Algorithmus mit der Gradientenschritt-Adaption und der Lernrate α_n für mehrere Prototypen pro Klasse. Es folgt eine Grafik für drei Prototypen pro Klasse.

[28] Vgl. [16], S. 28ff. und S. 36

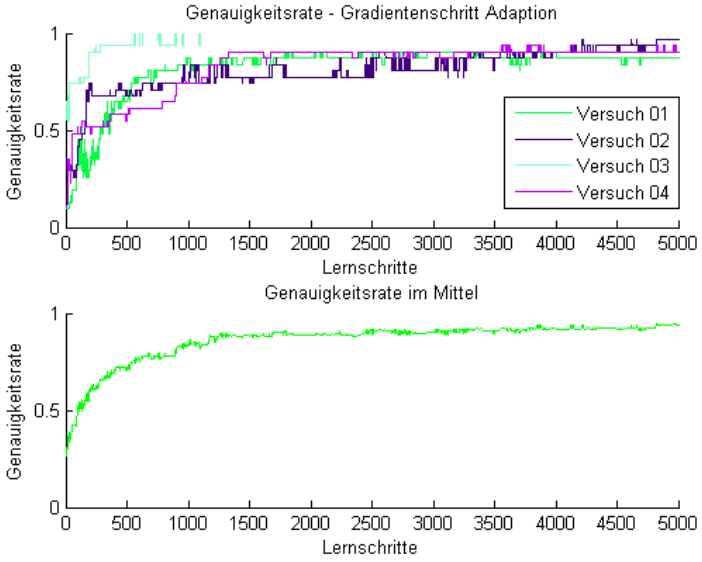

Abbildung 3.4: Genauigkeitsrate für den GLVQ- Algorithmus mit 5000 Lernschritten, Lernrate α_n und 3 Prototypen pro Klasse bzgl. Gradientenschritt- Adaption.

Man sieht deutlich das bessere Verhalten. Wir beschränken uns jedoch wieder auf einen Prototypen, da mit diesem schon gute Resultate erzielt wurden.

Die Versuche sind aufgrund der Abhängigkeit von den im Abschnitt 1 angesprochen Realisierungen von Zufallsvariablen und aufgrund des online Lernens ebenfalls Zufallsvariablen. Man kann also auch einen Erwartungswert schätzen. Aus [18] ist bekannt, dass man den Erwartungswert als Mittelwert mit m Stichproben schätzen kann. In diesem Beispiel wurde sich für $m = 10$ entschieden, da man aufgrund der Kreuzvalidierung zu 40 Versuche gelangt und da sich das Problem generell gut lernen lässt, wie aus Abbildung 3.3 ersichtlich. Die Parameter sind dabei wieder $N = 5000$ und es wird die Lernrate α_n verwendet. Man erhält folgende Grafik.

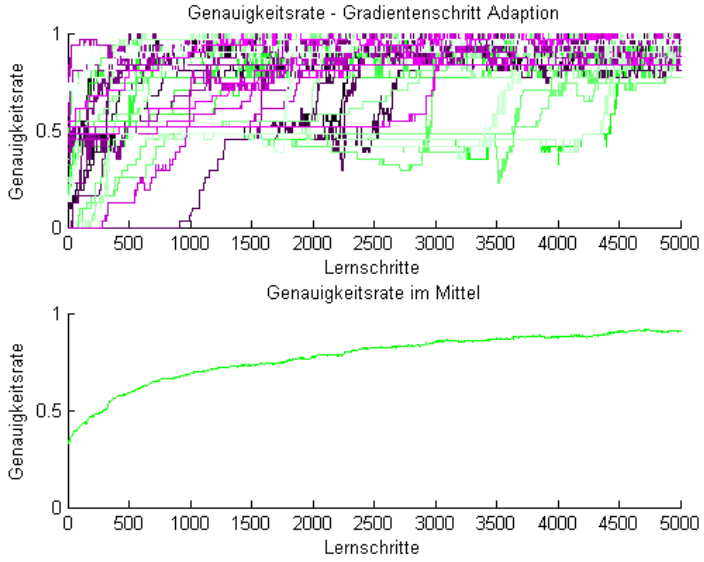

Abbildung 3.5: Erwartungswert der Genauigkeitsrate für den GLVQ mit Gradientenschritt- Adaption. Die 40 Versuche spiegeln sich in den Farben des oberen Bildes wieder.

Das Label für die 40 Versuche wurde aus Übersichtlichkeitsgründen weggelassen. Ferner steckt die Aussage im unteren Bild der Grafik. Diesem ist gut zu entnehmen, dass die Genauigkeitsrate bis auf kleine Schwankungen monoton bis zu einem bestimmten Niveau steigt. Dies wiederum ist für ein gut zu klassifizierendes und konvergentes Verfahren zu erwarten.

4 GLVQ unterstützt durch Newton [22]

Im vorangegangenen Kapitel wurde die Adaption durch (3.8) und (3.9) als Gradientenschritt-Adaption bezeichnet. In diesem Abschnitt wollen wir über die Newtonformel

$$x_{k+1} = x_k + \frac{F(x_k)}{F'(x_k)} \tag{4.1}$$

mit x_k als die zu adaptierenden Größen und $F(x)$ als die gegeben Funktion, zu einer Adaption gelangen, die wir als Newtonschritt- Adaption bezeichnen wollen. Die Newtonformel findet in der Numerik ihre Verwendung als Basisverfahren zur Lösung von nichtlinearen Gleichungssystemen der Art $F(x) = 0$[29]. Zur Motivation wollen wir erst die hinter der Newtonschritt- Adaption stehende Idee beleuchten. Denn diese ist der Anstoß zu dieser Arbeit.

4.1 Idee zur Verbesserung des GLVQ

Der Ursprung dieser Idee liegt in der Betrachtung der Konvergenzgeschwindigkeit der verwendeten Verfahren des GLVQ- Algorithmus. Schließlich konvergiert das Gradientenverfahren linear[30]. Das (n- dimensionalen) Newton- Verfahren konvergiert jedoch quadratisch[31]. Die Idee ist nun wie folgt, als Frage, formuliert:

Kann man ein Adaption der Prototypen für den GLVQ- Algorithmus über die Newtonformel (4.1) herleiten, so dass dieser Algorithmus schneller konvergiert?

Zu erwarten ist dann ein GLVQ- Algorithmus, der etwas aufwendiger in der Berechnung ist, aber durch die quadratische Konvergenz des Newton- Verfahrens schneller konvergiert, was den zusätzlichen Aufwand ausgleicht.

4.2 Herleitung der Adaption über den Newtonschritt

Die Grundlage bildet wieder die Energiefunktion (3.4) und deren Ableitungen (3.5) und (3.6). Es gilt nun die Newtonformel (4.1) geeignet zu interpretieren. Da wir mit dem GLVQ- Algorithmus Prototypen adaptieren, gilt $x_k = \omega_k^{\pm}$ mit $\omega_k^{\pm} \in \Omega$. Eine Anwendung des Newton- Verfahrens ist die Approximation der Nullstellen der Funktion $F(x)$. Weiterhin wollen wir die Energiefunktion minimieren und die notwendige Bedingung eines Ex-

[29] Vgl. [22], S.193
[30] Vgl. [22], S. 69 und S. 80
[31] Vgl. [22], S.191f. und S. 194f.

tremum der Funktion $E(x^*)$ im Punkt x^* ist $\nabla_\xi E(x^*) = 0$[32]. Also setzen wir $F(x) = \frac{\partial E}{\partial \omega^\pm}$ und gelangt zu folgender Adaption

$$\omega_{k+1}^\pm = \omega_k^\pm \pm \eta_k \cdot \frac{\left(\frac{\partial E}{\partial \omega^\pm}\right)}{\left(\frac{\partial^2 E}{\partial \omega^\pm \partial \omega^\pm}\right)} \tag{4.2}$$

mit der Lernrate η_k. Die Division der Vektoren erfolgt dabei komponentenweise. Für weiterführende Betrachtungen benötigt man die zweite Ableitung der Energiefunktion E. Da man wieder nach ω^+ und ω^- ableitet, gibt es vier zu betrachtende Fälle. Die reinen Terme der Ableitungen sind:

$$\begin{aligned}
\frac{\partial E}{\partial \omega^+ \partial \omega^+} &= \frac{-2d^-(d^+ - d^-)}{(d^+ + d^-)^4} \cdot \left(\frac{\partial d^+}{\partial \omega^+}\right)^2 + \frac{d^-}{(d^+ + d^-)^2} \cdot \underbrace{\frac{\partial d^+}{\partial \omega^+ \partial \omega^+}}_{=2} \\
&= \frac{2d^-}{(d^+ + d^-)^3} \left((d^+ + d^-) - \left(\frac{\partial d^+}{\partial \omega^+}\right)^2 \right) \\
\frac{\partial E}{\partial \omega^- \partial \omega^-} &= \frac{-2d^+(d^+ - d^-)}{(d^+ + d^-)^4} \cdot \left(\frac{\partial d^-}{\partial \omega^-}\right)^2 + \frac{d^+}{(d^+ + d^-)^2} \cdot \underbrace{\frac{\partial d^-}{\partial \omega^- \partial \omega^-}}_{=2} \\
&= \frac{2d^+}{(d^+ + d^-)^3} \left((d^+ + d^-) - \left(\frac{\partial d^-}{\partial \omega^-}\right)^2 \right)
\end{aligned}$$

Die gemischten Terme sind:

$$\begin{aligned}
\frac{\partial E}{\partial \omega^+ \partial \omega^-} &= \frac{\frac{\partial d^-}{\partial \omega^-}(d^+ + d^-)^2 - 2(d^+ + d^-)\frac{\partial d^-}{\partial \omega^-}}{(d^+ + d^-)^4} \cdot \frac{\partial d^+}{\partial \omega^+} + 0 \\
&= \frac{\frac{\partial d^-}{\partial \omega^-} \cdot \frac{\partial d^+}{\partial \omega^+}}{(d^+ + d^-)^3} \cdot \left((d^+ + d^-) - 2 \right) \\
&= \frac{\partial E}{\partial \omega^- \partial \omega^+}
\end{aligned}$$

Im Abschnitt 3 wurde festgestellt, dass man durch Beschränkung auf N Lernschritte lediglich zu einer Approximation der zu erlernenden Funktion gelangt. Da wir keine exakte Lösung anstreben, können wir weitere Vereinfachungen vornehmen um Rechenaufwand zu sparen. Deshalb vereinbaren wir $\frac{\partial E}{\partial \omega^+ \partial \omega^-} = \frac{\partial E}{\partial \omega^- \partial \omega^+} = 0$. Man kann in diesem Sinne noch etwas mehr approximieren, in dem man die zweite Ableitung durch den rückwärtigen Differentialquotienten ersetzt.

Definition 4.1 (Rückwärtiger Differentialquotient) Sei $y_i = f(x_i)$ der Funktionswert der Funktion f an der Stelle x_i. Dann lässt sich die Ableitung der Funktion f wie folgt ap-

[32] Vgl. [12], S. 7

proximieren:

$$DQ(f(x_i)) = \frac{y_i - y_{i-1}}{x_i - x_{i-1}}$$

Man nennt $DQ(f)$ den rückwärtigen Differentialquotienten bzgl. der Funktion f.

Diese Definition wollen wir für die zweite Ableitung der Energiefunktion nutzen:

$$\frac{\partial^2 E}{\partial \omega^{\pm} \partial \omega^{\pm}} = \frac{\frac{\partial E_t}{\partial \omega^{\pm}} - \frac{\partial E_{t-1}}{\partial \omega^{\pm}}}{t-(t-1)} = \frac{\partial E_t}{\partial \omega^{\pm}} - \frac{\partial E_{t-1}}{\partial \omega^{\pm}} \tag{4.3}$$

Hierbei ist $\frac{\partial E_t}{\partial \omega^{\pm}}$ die Ableitung der Energiefunktion im t-ten Lernschritt. Da man nun online lernt und damit die Trainingspunkte zufällig aus den Trainingsdaten zieht, kann es passieren, dass der Prototyp im Lernschritt $t-1$ und der Prototyp im t-ten Lernschritt zu unterschiedlichen Klassen gehören. Sei daher τ die Anzahl der Lernschritte die Vergangen sind als zuletzt ein Prototyp betrachtet wurde der der selben Klasse zugehörig ist wie der aktuelle Prototyp. Dann ändert sich (4.3) auf:

$$\frac{\partial^2 E}{\partial \omega^{\pm} \partial \omega^{\pm}} = \frac{\frac{\partial E_t}{\partial \omega^{\pm}} - \frac{\partial E_{t-\tau}}{\partial \omega^{\pm}}}{t-(t-\tau)} = \frac{\frac{\partial E_t}{\partial \omega^{\pm}} - \frac{\partial E_{t-\tau}}{\partial \omega^{\pm}}}{\tau} \tag{4.4}$$

Ersetzt man nun die in Gleichung (4.2) betrachtete zweite Ableitung durch den rückwärtigen Differentialquotienten der Gleichung (4.4), so erhält man:

$$\omega^{\pm}_{n+1} = \omega^{\pm}_n \pm \eta_n \cdot \tau \frac{\left(\frac{\partial E_t}{\partial \omega^{\pm}}\right)}{\left(\frac{\partial E_t}{\partial \omega^{\pm}} - \frac{\partial E_{t-\tau}}{\partial \omega^{\pm}}\right)} \tag{4.5}$$

Diese Adaption kann man wieder einzeln schreiben als

$$\omega^{+}_{n+1} = \omega^{+}_n + \eta_n \cdot \tau \frac{\left(\frac{\partial E_t}{\partial \omega^{+}}\right)}{\left(\frac{\partial E_t}{\partial \omega^{+}} - \frac{\partial E_{t-\tau}}{\partial \omega^{+}}\right)} \tag{4.6}$$

$$\omega^{-}_{n+1} = \omega^{-}_n - \eta_n \cdot \tau \frac{\left(\frac{\partial E_t}{\partial \omega^{-}}\right)}{\left(\frac{\partial E_t}{\partial \omega^{-}} - \frac{\partial E_{t-\tau}}{\partial \omega^{-}}\right)} \tag{4.7}$$

und wir bezeichnen diese Adaption als die Newtonschritt- Adaption. Damit ändert sich der im 3-ten Abschnitt angegebene Algorithmus **GLVQ** im Schritt (S4) und es gilt:

GLVQ:

⋮

(S4) adaptiere ω^{+}_{k+1} nach 4.6 und ω^{-}_{k+1} nach 4.7

⋮

4.3 Fortsetzung I des Iris- Blumen Beispiels

Setzten wir das Beispiel aus Abschnitt 3.2 mit dem angepassten GLVQ- Algorithmus fort. Man verwende wieder einen Prototypen pro Klasse und folgende Lernrate:

$$\eta_n = \frac{0.2 - 0.0001}{n^{\frac{1}{2}+0.005}}$$

Diese Lernrate ist zwar nicht die Günstigste für den GLVQ mit Gradientenschritt- Adaption, wie in Abschnitt 3.2 gesehen, aber es wird ja eine schnellere Konvergenz des Algorithmus mit der Newtonschritt- Adaption erwartet, so dass der Algorithmus trotz dieser Lernrate schnell konvergieren sollte. Die Anzahl der Lernschritte wurde auf $N = 5000$ begrenzt. Ferner wird zum Vergleich die Genauigkeitsrate des ersten GLVQ- Algorithmus aus Abschnitt 3 mit gezeichnet. Man gelangt zu folgender Grafik als Ergebnis.

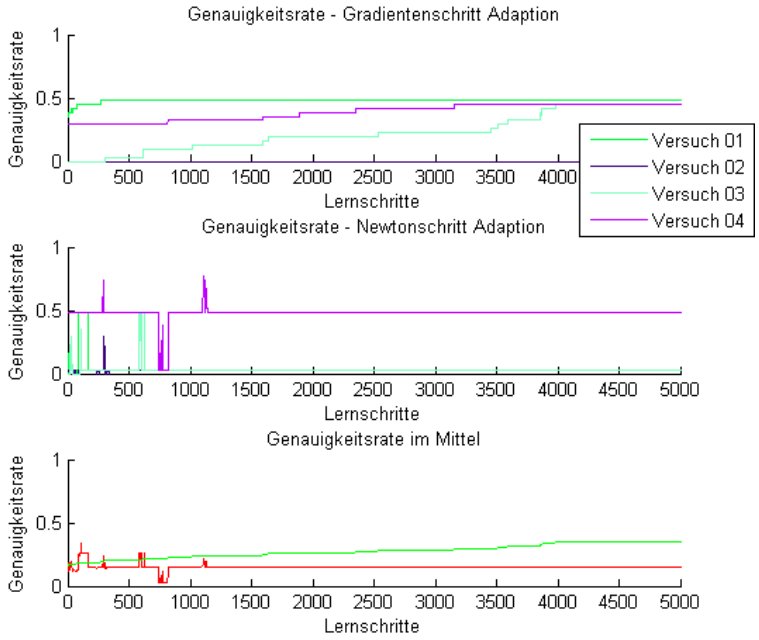

Abbildung 4.1: Genauigkeitsrate für den GLVQ mit Gradientenschritt- Adaption (oberstes Bild) und mit der Newtonschritt- Adaption (mittleres Bild). Das unterste Bild zeigt das Mittel der vier Versuche der Gradientenschritt - Adaption (grüner Graph) und der Newtonschritt- Adaption (roter Graph)

Aus dem mittleren und unteren Bild wird gegen der Erwartung ersichtlich, dass die Newtonschritt- Adaption lediglich am Anfang des Lernens ein wenig bessere Ergebnisse liefert, dann jedoch einstürzt und unbrauchbar wird. Das ist nicht das erwartete Ergebnis, da die Gradientenschritt- Adaption schon für diese Lernrate wesentlich besser Ergebnisse liefert und dazu stabiler konvergiert.

4.4 Analyse und Anpassungen der Newtonschritt-Adaption

Das Irisblumen Beispiel zeigt also, dass die Newtonschritt- Adaption in der Regel schlechter ist als die Gradientenschritt- Adaption. Aber woran liegt das? Nun dies wird der vorliegende Abschnitt klären.

Betrachten wir noch einmal Gleichung (4.5). Speziell wollen wir dem Nenner der Zuwachsrate betrachten, also den rückwärtigen Differentialquotienten $\left(\frac{\partial E_t}{\partial \omega^{\pm}} - \frac{\partial E_{t-\tau}}{\partial \omega^{\pm}}\right)$. Was passiert nun beim Lernen mit diesem Nenner? Es ist leicht einzusehen, dass, wenn der Algorithmus konvergiert, es einem gewissen Lernschritt gibt, sei dieser o.B.d.A. r, so dass sich $\frac{\partial E_r}{\partial \omega^{\pm}}$ und $\frac{\partial E_{r-\tau}}{\partial \omega^{\pm}}$ nicht mehr wesentlich voneinander unterscheiden. Sei dieser Unterschied zum Beispiel $\left(\frac{\partial E_r}{\partial \omega^{\pm}} - \frac{\partial E_{r-\tau}}{\partial \omega^{\pm}}\right) = 0.0001 = 1 \cdot 10^{-}3$. Damit steht in Gleichung (4.5) im r-ten Lernschritt

$$\omega^{\pm}_{r+1} = \omega^{\pm}_r \pm \eta_r \cdot \tau \cdot 10^3 \frac{\left(\frac{\partial E_t}{\partial \omega^{\pm}}\right)}{1}$$

Damit wird die Lernrate natürlich stark verändert und es folgt die beobachtete Instabilität des Verfahrens.

Numerisch gesehen folgt also die Instabilität aus der durch $\left(\frac{\partial E_t}{\partial \omega^{\pm}} - \frac{\partial E_{t-\tau}}{\partial \omega^{\pm}}\right)$ einhergehenden Stellenauslöschung[33].

Es stellt sich die Frage, ob man dies durch eine andere Darstellung von (4.5) vermeiden kann? Kürzen wir den Bruch mit $\frac{\partial E_t}{\partial \omega^{\pm}}$ und gelangen zu folgender Darstellung:

$$\omega^{\pm}_{n+1} = \omega^{\pm}_n \pm \eta_n \cdot \tau \frac{1}{\left(1 - \left(\frac{\frac{\partial E_{t-\tau}}{\partial \omega^{\pm}}}{\left(\frac{\partial E_t}{\partial \omega^{\pm}}\right)}\right)\right)} \tag{4.8}$$

Hierbei gilt es zu beachten, dass die verwendeten Einsen als Vektoren entsprechender Dimension zu verwenden sind. Es tritt jedoch im Nenner noch der Term $\frac{\frac{\partial E_{t-\tau}}{\partial \omega^{\pm}}}{\left(\frac{\partial E_t}{\partial \omega^{\pm}}\right)}$ auf. Und wenn wir diesen Komponentenweise dividieren, erhalten wir ein „Richtungsmischmasch“, da im Lernschritt $(t-\tau)$ ein von im Lernschritt t verschiedener Trainingspunkt $x_{t-\tau} \neq x_t$ gezogen werden kann, beide aber zur selben Klasse gehören, also $(x_{t-\tau}, i), (x_t, i) \in X_L$ gilt. Da in den Ableitungen die Richtung mit auftaucht, siehe Gleichung (3.5) und (3.6), erhält man also irgendeine Richtung, die aus der komponentenweisen Division der Richtungen hervor geht. Wir wollen jedoch, wie im Abschnitt 3 erklärt, den Prototypen in Richtung $(x_t - \omega^{\pm}_t)$ verschieben. Ferner arbeitet man mit Vek-

[33] Vgl. [22], S. 22

toren die durch Richtung und Länge charakterisiert sind[34]. Halten wir also die Richtung fest und lernen bzgl. der Länge $\|\cdot\|$ eines Vektors, so ergibt sich folgende Adaption:

$$\omega_{n+1}^{\pm} = \omega_n^{\pm} \pm \eta_n \cdot \frac{\tau}{\left(1 - \left(\frac{\left\|\frac{\partial E_{t-\tau}}{\partial \omega^{\pm}}\right\|}{\left\|\frac{\partial E_t}{\partial \omega^{\pm}}\right\|}\right)\right)} \cdot \left(x_n - \omega_n^{\pm}\right) \tag{4.9}$$

Hier ist die Eins als natürliche Zahl zu interpretieren. Ferner lässt sich diese Gleichung wieder einzeln als

$$\omega_{n+1}^{+} = \omega_n^{+} + \eta_n \cdot \frac{\tau}{\left(1 - \left(\frac{\left\|\frac{\partial E_{t-\tau}}{\partial \omega^{+}}\right\|}{\left\|\frac{\partial E_t}{\partial \omega^{+}}\right\|}\right)\right)} \cdot \left(x_n - \omega_n^{+}\right) \tag{4.10}$$

$$\omega_{n+1}^{-} = \omega_n^{-} - \eta_n \cdot \frac{\tau}{\left(1 - \left(\frac{\left\|\frac{\partial E_{t-\tau}}{\partial \omega^{-}}\right\|}{\left\|\frac{\partial E_t}{\partial \omega^{-}}\right\|}\right)\right)} \cdot \left(x_n - \omega_n^{-}\right) \tag{4.11}$$

schreiben. Damit lässt sich der Algorithmus **GLVQ** im Schritt (S4) wie folgt anpassen

GLVQNA:
⋮
(S4) adaptiere ω_{k+1}^{+} nach 4.10 und ω_{k+1}^{-} nach 4.11
⋮

Nennen wir diesen Algorithmus **GLVQNA** was für GLVQ- Newtonschritt- Adaption steht. Was passiert nun, wenn die Prototypen aufgrund der Konvergenz des Algorithmus sich ähnlich werden? Nun auch in (4.9) wird der Nenner $1 - \left(\frac{\left\|\frac{\partial E_{t-\tau}}{\partial \omega^{\pm}}\right\|}{\left\|\frac{\partial E_t}{\partial \omega^{\pm}}\right\|}\right)$ instabil, wenn sich die Prototypen nicht wesentlich unterscheiden. Man bekommt also die Instabilität nicht durch geeignete Umformungen weg. Betrachten wir dazu noch ein Beispiel.

4.5 Fortsetzung II des Iris- Blumen Beispiels

Setzten wir das Beispiel aus Abschnitt 3.2 mit dem neuen GLVQ- Algorithmus **GLVQNA**, das heißt mit der neuen Adaptionsregel, fort. Man verwende wieder einen Prototypen pro Klasse und folgende Lernrate:

$$\eta_n = \frac{0.2 - 0.0001}{n^{\frac{1}{2}+0.005}}$$

Die Anzahl der Lernschritte wurde als erstes auf $N = 1000$ begrenzt. Ferner wird zum Vergleich wieder die Genauigkeitsrate des ersten GLVQ- Algorithmus mit Gradientenschritt-

[34] Vgl. [9], S. 66

Adaption aus Abschnitt 3 mit gezeichnet. Man gelangt zu dem folgender Grafik als Ergebnis.

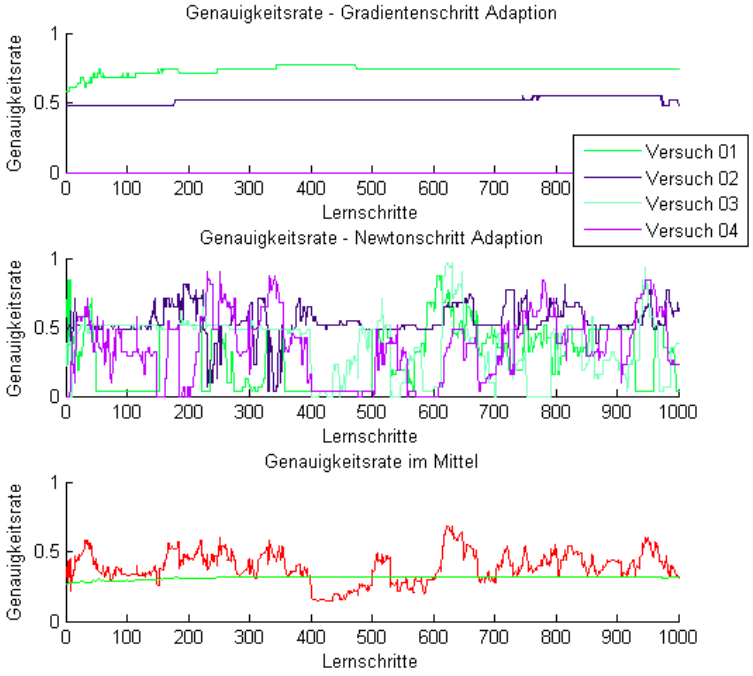

Abbildung 4.2: Genauigkeitsrate für den GLVQ- Algorithmus mit 1000 Lernschritten und 4 Versuchen bzgl. der Gradientenschritt- Adaption (oberstes Bild) und bzgl. der angepassten Newtonschritt- Adaption (mittleres Bild). Das unterste Bild zeichnet das Mittel der vier Versuche der Gradientenschritt - Adaption (grüner Graph) und der angepassten Newtonschritt- Adaption (roter Graph)

Aus der Grafik ist gut zu erkennen, dass gerade, wenn man schlecht initialisiert, siehe Versuch 04, die angepasste Newtonschritt- Adaption schnell zu besseren Ergebnissen als die Gradientenschritt- Adaption für die verwendete Lernrate führt. Da sich jedoch, wie schon festgestellt, auch die angepasste Newtonschritt- Adaption nicht stabil verhält, kommt es immer wieder bei relativ guter Genauigkeitsrate der angepassten Newtonschritt- Adaption zum Einsturz dieser Rate. Was dazu führt, dass kein stabiles Niveau erreicht wird. In der Grafik erkennt man dies gut am roten Graphen im unteren Bild. Noch besser ist das zu erkennen, wenn wir die Lernschritte N erhöhen. Für $N = 5000$ und mit der Lernrate η_n erhält man folgende Grafik.

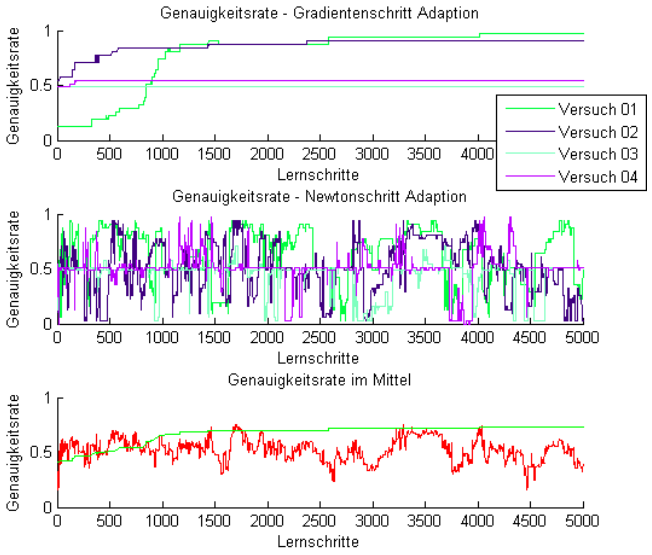

Abbildung 4.3: Genauigkeitsrate für den GLVQ- Algorithmus mit 5000 Lernschritten und 4 Versuchen bzgl. der Gradientenschritt- Adaption (oberstes Bild) und bzgl. der angepassten Newtonschritt- Adaption (mittleres Bild). Das unterste Bild zeichnet das Mittel der vier Versuche der Gradientenschritt - Adaption (grüner Graph) und der Newtonschritt- Adaption (roter Graph)

Diese Grafik verdeutlicht, dass sich die Newtonschritt- Adaption auf kein Niveau einpendelt. Noch besser ist das für $N = 10000$ ersichtlich.

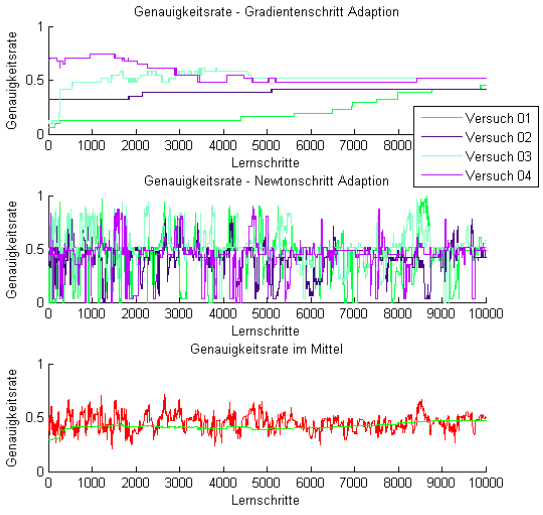

Abbildung 4.4: Genauigkeitsrate für den GLVQ- Algorithmus mit 10000 Lernschritten und 4 Versuchen bzgl. der Gradientenschritt- Adaption (oberstes Bild) und bzgl. der angepassten Newtonschritt- Adaption (mittleres Bild). Das unterste Bild zeichnet das Mittel der vier Versuche der Gradientenschritt - Adaption (grüner Graph) und der Newtonschritt- Adaption (roter Graph)

Mehrere Prototypen ändern an diesem Verhalten nichts.

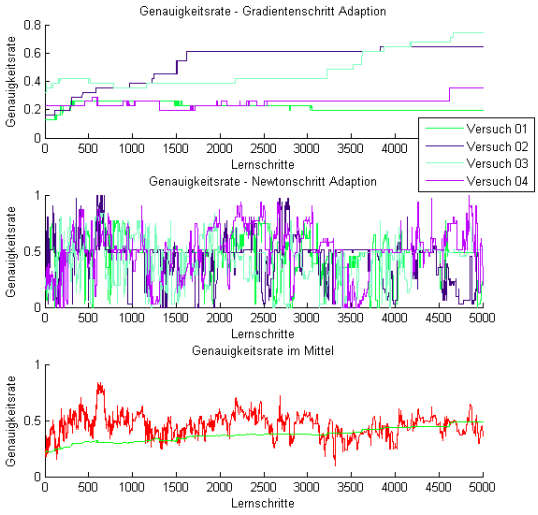

Abbildung 4.5: Genauigkeitsrate für den GLVQ- Algorithmus mit 5000 Lernschritten, 4 Versuchen und 3 Prototypen pro Klasse bzgl. der Gradientenschritt- Adaption (oberstes Bild) und bzgl. der angepassten Newtonschritt- Adaption (mittleres Bild). Das unterste Bild zeichnet das Mittel der vier Versuche der Gradientenschritt - Adaption (grüner Graph) und der Newtonschritt- Adaption (roter Graph).

Um sicher zu gehen, dass sich dieses Verhalten fortsetzt, wird noch mal mit 40 Versuchen, $N = 5000$ und einem Prototypen pro Klasse gelernt. Man erhält folgende Grafik.

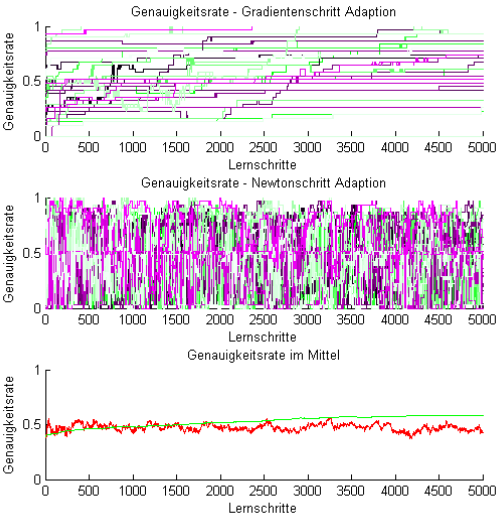

Abbildung 4.6: Genauigkeitsrate für den GLVQ- Algorithmus mit 5000 Lernschritten und 40 Versuchen bzgl. der Gradientenschritt- Adaption (oberstes Bild) und bzgl. der angepassten Newtonschritt- Adaption (mittleres Bild). Das unterste Bild zeichnet das Mittel der 40 Versuche der Gradientenschritt - Adaption (grüner Graph) und der angepassten Newtonschritt- Adaption (roter Graph)

Wie in Abbildung 3.5 wird das Label aus Übersichtlichkeitsgründen weggelassen. Aus der hier vorliegenden Grafik ist zu erkennen, dass sich das Verhalten bzgl. eines festen Niveaus, wie schon zu erwartet, nicht ändert. Das führt nun dazu, dass man immer ein N finden kann, so dass der GLVQ- Algorithmus mit der Gradientenschritt- Adaption bessere Ergebnisse liefert als für die Newtonschritt- Adaption.

5 Zusammenfassung

Fassen wir die Ergebnisse zusammen. Im Allgemeinen wird der Algorithmus **GLVQ** dem Algorithmus **GLVQNA** vorzuziehen sein. Sollte man sich jedoch mit einem Problem beschäftigen, welches fast sicher oder oft eine schlechte Initialisierung der Prototypen besitzt, so ist eine Kopplung der Algorithmen **GLVQ** und **GLVQNA** denkbar. Man bestimme sich dazu ein Level bis zu dem man mit dem Algorithmus **GLVQNA** lernt und verwende ab diesem Level den Algorithmus **GLVQ**, welcher die gelernten Prototypen des Algorithmus **GLVQNA** als Initialisierung verwendet.

Literaturverzeichnis

[1] *Chinesische Ziffern*. http://www.compassmuseum.com/images/miscell/chinese_figures_gr.jpg.

[2] *Iris- Blume*. http://w3.uniroma1.it/chemo/image/iris-flower.jpg.

[3] *Iris Data Set*. http://archive.ics.uci.edu/ml/datasets/Iris.

[4] *rand*. http://www.mathworks.com/help/techdoc/ref/rand.html.

[5] ALPAYDIN: *Maschinelles Lernen*. Oldenburg Wissenschaftsverlag, München, 1 Auflage, 2008. Übersetzt von S. Linke.

[6] DAMMEIER: *Das BMP Format*. http://eschumacher.de/Info/I_Skripte/bmp-format.pdf.

[7] DEISER: *Einführung in die Mengenlehre*. Springer-Verlag, Berlin, Heidelberg, 2 Auflage, 2002.

[8] DREISEITL: *Allgemeines zum Gradientenabstiegsverfahren*. http://staff.fh-hagenberg.at/sdreisei/Teaching/WS2001-2002/PatternClassification/graddescent.pdf. Vorlesungsskript.

[9] EISENREICH: *Lineare Algebra und analytische Geometrie*. Verlag Harri Deutsch, 1 Auflage. Nachdruck.

[10] FICHTENHOLZ: *Differential- und Integralrechnung 1*, Band 1. Wissenschaftlicher Verlag Harri Deutsch, 14 Auflage, 1997, 2006.

[11] FRANKLIN, 1789. Briefe an Leroy.

[12] GEIGER und KANZOW: *Numerische Verfahren zur Lösung unrestringierter Optimierungsverfahren*. Springer-Verlag, 1 Auflage, 1999.

[13] HAMMER: *Neural Computation*. TU- Claustal, 2008. Skript.

[14] KLENKE: *Wahrscheinlichkeitstheorie*. Springer-Verlag, Berlin, Heidelberg, 2 Auflage, 2008, 2006.

[15] KOHONEN: *Self-Organizing Maps*. Springer-Verlag, Berlin, Heidelberg, 3 Auflage, 1995, 1997, 2001.

[16] KUSHNER und CLARK: *Stochastic Appproximation Methods for Constrained and Unconstrained Systems*, Band 26. Springer-Verlag, New York, 1 Auflage, 1978.

[17] MAX-PLANCK-GESELLSCHAFT ZUR FÖRDERUNG DER WISSENSCHAFTEN: *Max-PlanckForschung: das Wissenschaftsmagazin der Max-Planck-Gesellschaft*. Max-Planck-Gesellschaft zur Förderung der Wissenschaften, Seite 15, 2007.

[18] MOSLER und SCHMID: *Wahrscheinlichkeitsrechnung und schließende Statistik*. Springer-Verlag, 3 Auflage, 2008, 2006, 2004.

[19] PHILIPP: *stochastischer Gradientenabstieg*. http://flashpixx.de/2010/02/stochastischer-gradientenabstieg/.

[20] RUDIN: *Analysis*. Oldenburg Wissenschaftsverlag, 4 Auflage, 2009.

[21] SATO und YAMADA: *Generalized Leraning Vector Quantization*. In: TOURETZKY, HASSELMO und MOZER (Herausgeber): *Advances in Neural Information Processing Systems 8*, Seiten 423 – 429. Mit Pr, 1996.

[22] SCHWETLICK und KRETZSCHMAR: *Numerische Verfahren für Naturwissenschaftler und Ingenieure*. Fachbuchverlag Leipzig, 1 Auflage, 1991.

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Mittweida, 26.11.2010